Introduction

Team Name: 3J & A

Team Members: Jamie Laurin, Jazmin Lor, Audrey Soares, Jake Walker

Application Title: MoneyManager

Description: A personal budget tracker where you can manage and track your spendings and savings. This application takes in your income amounts and based on a monthly basis, you may add in your spendings, bills, and subscriptions to see how much you may have left over. From there on, you may accumulate them into savings or pay off any loans or debt.

Function Requirement Specification:

- Profile name (non-sensitive)
- Email address (sensitive)
- Password (sensitive)
- Monthly budget (sensitive)
- Budgets by categories (sensitive)
- Expenses (sensitive)

Type of Program: Web-based Application

Development Tools:

- Javascript
- HTML/CSS
- Git/Github
- IntelliJ IDEA

Requirements

Security and Privacy Requirements: Login usernames and passwords will be stored securely using software encryption. Budgets and expenses will also be stored, but solely only for the convenience of the user. This application will also require a personal 4-digit security code in order to access their MoneyManager account (due to time constraints, this feature is not included in v1.0). This is to prevent anyone from logging in if their phone or login data has been stolen. We will be using GitHub Projects to manage our tasks and issues. Our application will be updated every now and then with patch updates and will also comply with various regulations. As a plan to keep track of the security flaws that may arise, the team members will perform continuous threat monitoring on the application.

Quality Gates (or Bug Bars):

Privacy Bug Bar

- *Critical*–a scenario in which a lawsuit against a business may arise alleging financial damage.
    - *Lack of notice and consent.* The application transfers sensitive data from the user's system without notice and consent in the UI prior to transfer.
    - *Lack of user control.* Ongoing transfer of sensitive data from the user's system without the ability within the UI for the user to stop the transfer.
    - *Lack of data protection.* The application stores sensitive data without encryption and allows users to access the data without an authentication mechanism.
- *Important*–a scenario in which a business may receive backlash from the public.
    - *Lack of notice and consent.* The application transfers non-sensitive data from the user's system without notice and consent in the UI prior to transfer.
    - *Lack of data protection.* The application allows users to access non-sensitive data without an authentication mechanism.
- *Moderate*–a scenario in which a business may receive criticism from privacy advocates.
    - *Lack of user control.* The application stores sensitive data locally as hidden metadata without any means for a user to remove the metadata.
- *Low*–a scenario in which some users may raise questions or voice concerns.

- *Lack of notice and consent.* The application stores sensitive data locally as hidden metadata without notice or consent.

Security Bug Bar

- *Critical*–a security vulnerability that could cause severe damage.
  - *Elevation of privilege.* An attacker executes arbitrary code or obtains more privilege than authorized.
- *Important*–a security vulnerability that could cause significant damage.
  - *Denial of service.* An attacker sends a small amount of data to disrupt services.
  - *Information disclosure.* An attacker can locate and read sensitive information as well as system information that was not intended to be exposed.
  - *Tampering.* An attacker is able to permanently modify user data in any scenario.
  - *Security features.* An attacker breaks or bypasses any security feature provided.
- *Moderate*–a security vulnerability that could cause moderate damage.
  - *Tampering.* An attacker is able to permanently modify user data in a specific scenario.
- *Low*–a security vulnerability that could cause low damage.
  - *Tampering.* An attacker is able to temporarily modify user data in a specific scenario.

Risk Assessment Plan for Security and Privacy:

- Identifying the Project and Key Privacy Contacts
  - The name of our project is MoneyManager and our app will be projected to launch by the date Assignment 5 is due. Everyone on our team will be responsible for privacy.
- Privacy Impact Rating
  - P3
- Privacy/Security Analysis

  - Data that we will be storing and collecting will be what the user manually inputs, such as their budgets, expenses, and subscriptions they have. We are still unsure if we are going to connect bank accounts, but if so, then this increases the privacy and security measures (due to time constraints, this feature is not included in v1.0). However, if this method is used, the user's bank information will not be shared and will only be used for tracking their spendings and earnings. This data will then be reflected on the user's MoneyManager account. We will have consent once the user agrees to the terms and conditions once they connect their bank account(s). If a bank account is not connected, then gaining their consent will be done once registration happens. Our terms and conditions will be accessible to the public on the homepage. Although organizations may not control our features, users will have control over their own information and features. This includes their data, information, and whatever they choose to manually input. We will prevent unauthorized access by only strictly allowing the users themselves to log into their account on the application. To add more security, once their login details are entered, users have the option to enable two-factor authentication (2FA) to add another layer of protection in addition to their password. In the case that the software is involved in a privacy incident, please contact Jazmin Lor at lorj@hawaii.edu.

Design

Design Requirements:

- Password encryption/protection
  - Passwords will be stored using software encryption
- User information
  - Usernames will be stored using software encryption
  - User information will only be visible when the user logs in with their username, password, and 2FA if enabled

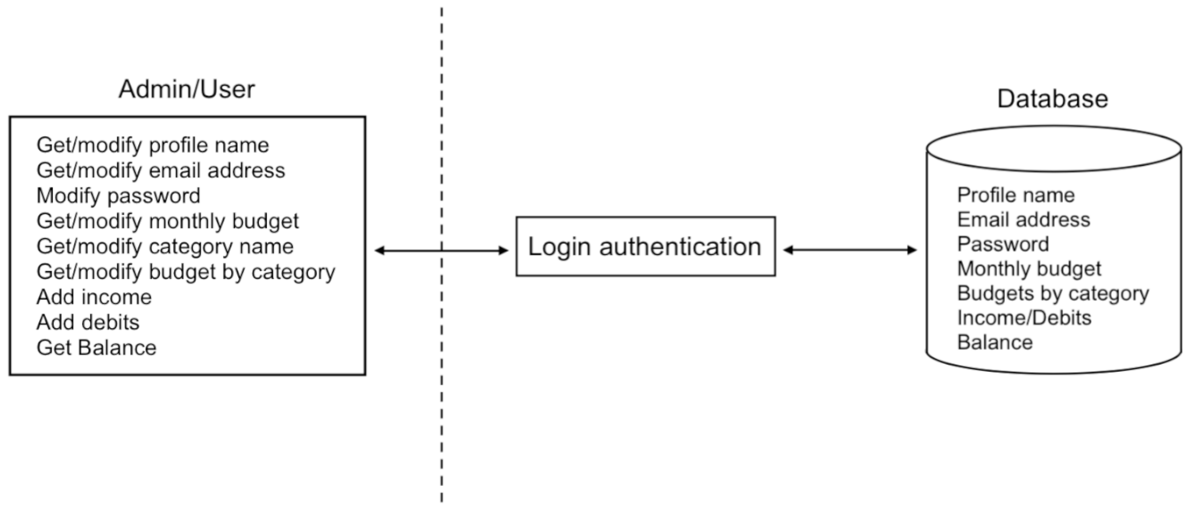
    - Changes to user information can only be made when the user is logged into their account
- Database
    - User’s profile name
    - User’s email address
    - User’s password
    - User’s budgets
    - User’s expenses

Attack Surface Analysis and Reduction:

Privilege Levels

- User
    - Read and write their user information
        - Reads their user information when logged in
        - Makes changes to their information when logged in
        - Get/add/modify user budget and/or expenses

Threat Modeling:



Potential threats:

- *Denial of service.* An attacker sends a small amount of data to disrupt services.

- *Information disclosure.* An attacker locates and reads user data.
- *Tampering.* An attacker modifies user data.
- *Security features.* An attacker breaks or bypasses the login authentication.

Implementation

Approved Tools:

Tool/Framework	**Version**
IntelliJ IDEA	2023.1.2
HTML	Living Standard–Last Updated 7 June 2023
CSS	CSS3
JavaScript	ECMAScript 2022
Bootstrap	5.3.0
React	18.2.0
Meteor	2.12.0
MongoDB	6.0.6
Git	2.41.0

Deprecated/Unsafe Functions:

React

- ReactDOM.render - was used to create a root to render or unmount. React 18 doesn't work with it, and using it will run your app in React 17 mode.
  - Alternatively, use createRoot
- ReactDOM.hydrate - was used to hydrate a server-rendered application. React 18 doesn't work with it and using it will run your app in React 17 mode.
  - Alternatively use hydrateRoot

- renderToString - used to render a React tree to an HTML string.
    - Alternatively use renderToPipeableStream or renderToReadableStream

Meteor

- allow/deny - are used to specify whether a client is allowed to run MongoDB queries to write directly to the database. Meteor documentation strongly recommends avoiding these functions as they could potentially become hard to manage as the project grows over time.
    - Alternatively use Meteor.methods
- insecure - is a package that allows clients to run MongoDB queries to write to the database. This package is used for prototyping and should be removed before release.
    - To remove this package, go to the app directory and run ‘meteor remove insecure’
    - Alternatively use Meteor.methods
- autopublish - is a package that publishes all data to the clients. This package is used for prototyping and should be removed before release.
    - To remove this package, go to the app directory and run ‘meteor remove autopublish
    - Alternatively use Meteor.publish

Static Analysis:

Tool	**Version**
eslint	8.42.0

ESLint is a tool that looks through code as it is typed and notifies the programmer of any errors. These include spacing and indentation errors, as well as unreachable code or unused variables. Since code does not need to be executed for errors to be found and fixed, it is a very helpful tool for programmers to use. Using ESLint ensures that everyone is conforming to the same coding standards which makes our code easy to read and understand. Since everyone conforms to the same coding standards, merging code worked on by different people does not create a conflict. ESLint also offers suggestions for solutions to problems so that programmers can easily fix their code and continue coding.

ESLint may be used in a program by being enabled, or may be run as a plugin using Meteor. For example, it may be run using the command line code “meteor npm run lint”. This command line is used through Meteor. The pros of using it through Meteor is, as explained above, it allows you to see any grammatical or computational errors. These are errors or typos that you may not even know about, even if your application or program runs just fine. Just enabling it alone in the IDE will only give specific errors if your application or program cannot run appropriately.

So far, this tool has been extremely effective and helpful! We have had no merge conflicts and before each commit is done to the main branch, we are ensuring that ESLint is executed and cleared.

Verification

Dynamic Analysis:

Tool	**Version**
TestCafe	2.3.1

Testcafe is a free and open-source web-application tester. Throughout our application development, our code must consistently pass a series of tests to be considered as “passing”. Our tests consist of whether a web-page is displayed correctly and its functionality (ex. logging in and out of a user’s account).

Upon checking and merging our “issue” branches to the main branch, we are all ensuring we are checking if the badge is “passing” or “failing”. If it is passing, that means everything is working correctly. However, if it is failing, something is going wrong due to the new merge. We are then able to run TestCafe through Meteor on the IDE and identify what is wrong, or delete the merge and fix whatever is wrong before merging into the main again.

So far, after using TestCafe multiple times now, we have found it to be extremely useful. Similar to ESLint, it is able to tell us the errors (if any) and how to fix them. There are some instances where our badge would be “failing” for a while, but thankfully, with the help of our team members, we are able to get it back up and passing. It has been very successful so far.

Our web application is based on a template which means our goals in functionality will differ from that of our template. That means we did and will continue to face some difficulties when it comes to constructing new tests for our application. This will be tricky since our application is still in the development stage, and our tests may not be finalized until the end of our project. However, we found it to be very useful in our development process. It ensures that any user will be able to navigate through our application without major problems.

Here is an example of what our current (and a passing) badge is looking like:

Attack Surface Review:

Since our last update, there have not been any changes with our approved tools. Due to our short time frame and quick progression of our application, we have not found it necessary to update any software or tools. We have still been using the same approved tools and software, as well as their versions. There have also not been any new vulnerabilities reported. Nevertheless, this does not mean our project is susceptible to change in the future.

Fuzz Testing:

Brute force attack

One way that a malicious attacker would attempt to hack the system to obtain sensitive information from the user is to guess the user's password by trying out commonly used passwords. Unfortunately, our app currently does not have a feature where it enforces users to choose a strong password, i.e., a password that is at least eight characters, including upper and lowercase letters and at least one number. Due to the time constraint of this project, we are unsure if we will be able to incorporate this feature before the release, however, we are aware of this vulnerability and will address the issue if the app is ever released to the public.

If, by chance, a user's phone is stolen or the hacker obtains the user's username and password, there is a personal 4 digit PIN number that must be correctly entered in order to successfully log in. This PIN number will only allow numbers, a max of 4 digits, and must be correct. This is to basically add a second a second password just incase the user's actual password has been compromised. (Due to time constraints, this feature is not included in v1.0.)

Two-factor authentication

Our app incorporates two-factor authentication (2FA) for users who would like to have stronger protection against malicious attackers. Although 2FA is not required, if enabled, it becomes extremely difficult for an attacker to bypass the login screen. One way that an attacker would attempt to bypass the 2FA is to guess the correct 6-digit 2FA code. However, this requires guessing the correct 2FA code out of 1,000,000 possibilities as well as the correct user password which would make the probability of the attacker bypassing the login system extremely low.

Users are able to enable 2FA from the Security Settings page once logged in. We have tested at least 10 times to ensure that users are able to enable (or disable) 2FA and that the login screen works accordingly. From these tests, we have found that 2FA works as expected.

Static Analysis Review:

There have been no changes when committing any static analysis so far. We have been consistently using ESLint throughout the process of this project. If we ever need to commit to any changes in our repository, we first ensure it's free of errors. As soon as the command “meteor npm run lint” runs in our terminal, it will quickly notify us of any discrepancies in our program and tells us exactly which needed to be changed and why. We believe it's best to rid of all syntax errors as soon they’re found, so all errors in our project have been addressed. There was an exception in our program in which we had to suppress an ESLint error due to shadowing, where a local variable shared the same name as a variable in its scope. This was done by disabling ESLint just for that one line of code, but doing this did not harm our project in any way.

Dynamic Review:

We are still using Testcafe to test our application, to ensure pages are displayed correctly and functional. However, we’re still in the process of adjusting specific functionalities for some pages. That means we won’t be able to create tests for those specific functions. With that in mind, we are constantly going in and testing the web application ourselves for these functions. To inspect what's happening in the process, we are using the inspect tools from our web browser. Through the inspection tools, we are able to examine the cause of errors, such as invalid entries or if certain terms are not defined properly. A helpful tool we frequently used was MiniMongo, it allowed us to inspect our collections as we made changes on the client side, whether it be adding or editing entries. To ensure that we were collecting and updating the correct data for each entry.

Release

Incident Response Plan:

Privacy Escalation Team:

- Jazmin Lor - Escalation Manager @ lorj@hawaii.edu
  “*My job is to ensure my customer service team is providing the best service and the most accurate information that they are able to provide. If they are not able to answer a customer’s question, the inquiry will be forwarded to the appropriate department or spokesperson. I am also ensuring that every organization throughout the company is at its best with our top notch representatives. Not only do I assign my team to specific tasks and workloads, but I also ensure that all aspects of any escalations are resolved.* ”

- Jake Walker - Legal Representative @ jakesw@hawaii.edu
  “*In my line of work, I am responsible for resolving any legal concerns and questions. I enforce and ensure the company is following all legal and up-to-date policies.* ”

- Jamie Laurin - Public Relations Representative @ jlaurin8@hawaii.edu
  “*Similar to a social media manager, I am responsible for the company’s public image. I manage and promote on all social media platforms with my team, as well as rebranding and creating new advertisements. Another duty of my role is to maintain and expand our public relations outreach.* ”

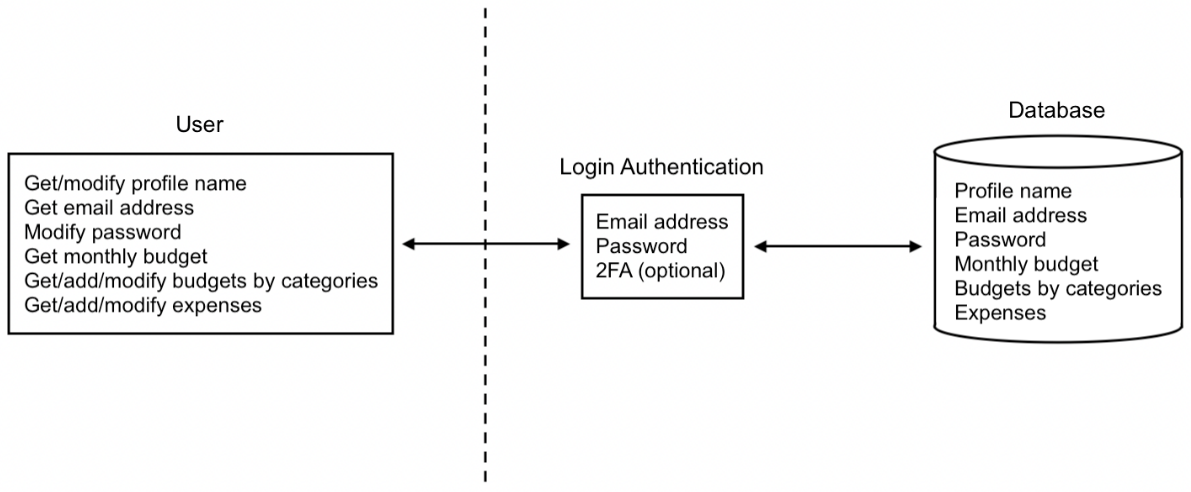
- Audrey Soares - Security Engineer @ amsoares@hawaii.edu
  “*My role is to monitor and always come up with new ideas to patch and prevent further vulnerabilities. Performing, developing, and testing new security measures is the highlight of my job. I am also in charge of all patchworks, ensuring our program meets security requirements.* ”

In the case that there is an emergency, our team may be reached at 3JandA@gmail.com.

Process and Procedure:

When the first e-mail, call, or any form of communication of an issue is received, it will always be the escalation manager who is responsible for ensuring the issue will be, can, and is fully resolved. Their task will be to determine the extent and severity of the issue, as well as assigning a specialist to the issue. If needed, the legal and public relations representatives may step in and aid in resolving the issue further. If there are any needed changes or improvements to the system or software, the security engineer will step in and apply those changes.

Final Security Review:



Our team's threat model is the same as the original with only minor changes to the user's data.

The final static analysis of our code resulted in no errors, with the addition of suppressed errors which had no impact on our program.

Due to time constraints, our final dynamic analysis of our code was not as thorough as we wanted it to be. It passed all current tests which were present, but we lacked in-depth testcafe tests to test all of the functionalities of our web pages. We had to manually test our project multiple times for some of the features.

With further penetration testing for our application, we would grade the functionalities of our program to be a Passed FSR. Users have the option to secure their accounts with Two Factor Authentication, and only one account can be made per email. A user's expense and budget information, may not be modified or viewed by other users.

Certified Release & Archive Report:

The release for MoneyManager may be found here:

Released Version of MoneyManager

Version: 1.0.1

Summary of Features:

- Two-Factor Authentication
- Personal PIN Number (in progress)
- Track Spendings and Budget by Categories

Future Development Plans:

- Monthly Expense Reports, show transactional history of previous months
- User PIN-activated pages (in progress)

Installation:

- First Install Meteor
- You will also need to download Released Version of MoneyManager

- Once downloaded, cd into moneymanager/app directory and install libraries with

```
$ meteor npm install
```

- Then continue to run the system with:

```
$ meteor npm run start
```

- If all goes well, the application will appear at http://localhost:3000/

Once all the steps above have been completed, your installation of MoneyManger is complete! You can create an account, modify the security settings to your preference and start budgeting your wallet skillfully!